The 32nd Annual Conference of the Japanese Society for Artificial Intelligence, 2018

Neural Programmer-Interpreters の拡張と四則演算を用いる文章問題の解答

Solving Four Arithmetic Operation Problems and Word Algebra Problems by Neural Programmer-Interpreters

勝俣 翔太 *1　井上 克巳 *1*2
Shota Katsumata　Katsumi Inoue

*1東京工業大学 情報理工学院
Tokyo Institute of Technology, School of Computing

*2国立情報学研究所
National Institute of Informatics

In this paper we extend the arithmetic operations of the Neural Programmer-Interpreters (NPI). NPI is a recurrent and compositional neural network that learns to represent and execute programs. First, we enable NPI to execute not only the addition that NPI was originally possible but also the other three arithmetic operations, i.e. subtraction, multiplication and division. Then, we extend NPI by allowing it to share subprograms between tasks for improving learning efficiency. Next, we solve word algebra problems for elementary school-level mathematics which can be solved by using four arithmetic operations. For this purpose, we develop a converter that converts word algebra problems into mathematical expressions. This neural network is based on the Sequence-to-Sequence model with the attention mechanism. Using this neural network and NPI, we solve the data sets of word algebra problems and show that the accuracy of our method is better than the other existing methods.

1. はじめに

Programming by example (PBE) は実行したいタスクの入出力例からコンピューターが自動的にタスクを達成するためのプログラムを推論する技術である [9]．Neural Programmer-Interpreters (NPI) とは Reed らによって提案されたプログラムの実行を学習できる PBE の一手法であり、ニューラルネットワークを用いる [10]．学習後の NPI は実行すべきタスクと環境を情報として受け取り，そのタスクを達成するためのプログラムシーケンスを出力する．NPI は再帰構造とニューラルネットワークを用いることにより、タスクの長さに対する強力な汎化性能を持ち、高い精度でプログラムを出力できる．

ニューラルネットワークを使ってコンピューターにプログラムを理解させる研究は近年大きな成果を見せている．Graves らは NTM を開発した [1]．これはチューリングマシンの読み書きなどの動作を微分できる行列計算として定義し、外部メモリと LSTM コントローラを用いて、簡単なプログラムの実行を学習する技術である．Shu らはテキストを変換するプログラムを変換前と変換後のテキストのペアから導出する NPBE を提案している [3]．類似する手法として、Parisotto らは NSPS を提案している [5]．

本研究ではまず、NPI が可能であった加算の筆算に加えて、減算、乗算、除算のそれぞれの筆算を NPI ができるようにタスクを定義する．その際、タスク間でサブプログラムを共有可能にさせる拡張を行い、学習効率を向上させる．これは [10] で今後の課題とされていたことのうちの 1 つである．次に、四則演算を使って解くことのできる、小学生レベルの算数の文章問題を解くことを考える．このために、文章問題を数式表現へと変換する変換器を作成する．この変換器と NPI を用いて文章問題のデータセットを解き、他の既存手法と比較する．本研究では他の既存手法と違い、答えを求めるとともに NPI のプログラムシーケンスも得られる．すなわち、自然言語表現からプログラムを出力することができる．

連絡先: 勝俣 翔太，東京工業大学，shota@il.c.titech.ac.jp

2. NPI

NPI の構造を簡潔に表した図を 1 に示す．NPI は環境からの情報、現在のプログラム ID、その引数を入力として受け取り、プログラムの終了フラグ、次のプログラム ID、その引数をステップごとに出力する．プログラム ID は整数であり、ID のリストはユーザーによって与えられ、引数は整数 3 つのタプルである．環境からの情報は環境によって可変だが、本研究では環境に存在する 4 つのポインターが何を指しているかを LSTM-core に与える．学習を行う際はこれら入出力のペアを訓練データとし、テスト時には各ステップにおける出力を次ステップの入力として用いる．NPI はこの入力と出力のステップを繰り返し行うことで出力系列を生成し、NPI 全体の出力とする．図 2 は我々が定義した減算タスクを NPI が実行している様子を表している．NPI はプログラムの実行手順をステップごとに区切って学習することで、出力すべきプログラム長に対する強力な汎化性能を持つ．Cai らはこの汎化性能を完全にする改良を行なっており、本研究ではこの手法に基づいて拡張を行う [8]．

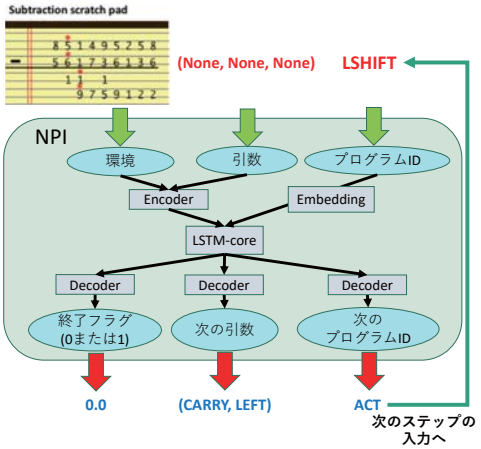

図 1: NPI の構造

3. NPI の拡張

3.1 モデル

NPI を四則演算に対応させるための拡張を行った．その際、環境の種類とエンコーダーを統一することでサブプログラムをタスク間で共通することを可能としている．環境には 4 つのポインターが存在し、NPI は各ポインターを移動させたり、ポインターの位置に任意の数字を書くことができる．また、そのポインターを通じて NPI に環境を観測させる．図 2 は拡張した NPI の減算における動作の様子を示している．

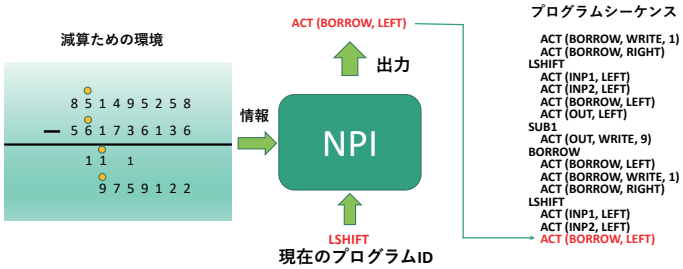

図 2: NPI の減算実行時の様子

3.2 タスクの定義について

NPI を四則演算に対応させるためには、環境とプログラムリストを定義する必要がある．加算については [10] と同じものであるが、減算については、桁上げに代わって桁借りを組み込む．これらを行うかどうかも NPI は学習することができ、桁上げの必要がない場合に 0 を桁上げしたりすることはない．以下では乗算と除算について説明する．

3.2.1 乗算

環境は図 3 の通りである．乗算タスクでは通常の筆算と同じように，大きく分けて 2 つのステップで処理を行う．まず、図 3 の左のように，最初に 1 桁同士の乗算部分を行う．次に，図 3 の右のように前ステップで出力された数字を合計していく．

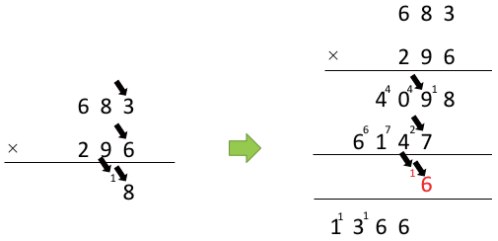

図 3: 乗算プログラム実行時の環境 (一部簡略化している)

3.3 除算

本研究では，乗算と減算を繰り返し実行することによって除算を実行する．その様子を図 4 に表す．図の上部では乗算，および減算の様子を表している．初めに，商を出力するべき欄の最左端に 9 を立てる．その後、出力した数字、割る数、割られる数とで乗算と減算を行う．その結果が負であれば乗算と減算の結果を消去し、次は 9 から 1 を引いて 8 を立てる，というようにして商を探索していく．図の下部では、立てた商が正しかった場合に次の桁へと移る処理の様子を表している．

3.4 サブプログラムの共有

本研究ではサブプログラムの共有によって学習効率を向上させている。図 5 は NPI が実際に出力する加算と減算のプログラムシーケンスの一部である。図中にある 2 つの LSHIFT は全く同じ動作をするプログラムである。このようなプログラムに同一のプログラム ID を与え、学習状況を共有させた。このような全てのプログラムに対して共有を行なった。

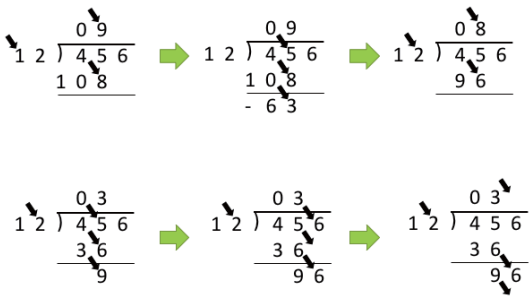

図 4: 除算プログラム実行時の環境 (一部簡略化している)

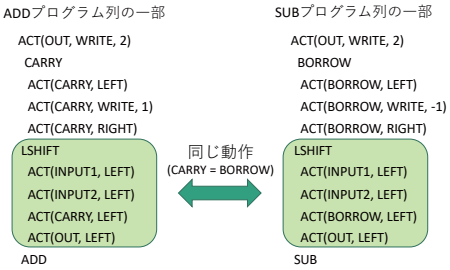

図 5: 加算と減算のプログラムシーケンスの一部

4. 文章問題への対応

4.1 概要

ここで扱う文章問題とは “Luke was trying to expand his game collection. He bought 4 games from a friend and bought 5 more at a garage sale. How many good games did he end up with?” のような，四則演算のみを 1 回または 2 回使った数式表現に変換できるものを指す．これをアテンションメカニズム [4] を組み込んだ Sequence-to-Sequence モデル [7] を基にした変換器を使って数式表現へと変換し (上の例であれば「4 + 5」)，それを NPI を使って解き，プログラムシーケンスを出力することを目的とする．一連の流れを図 6 に、変換器の図を 7 に示す．

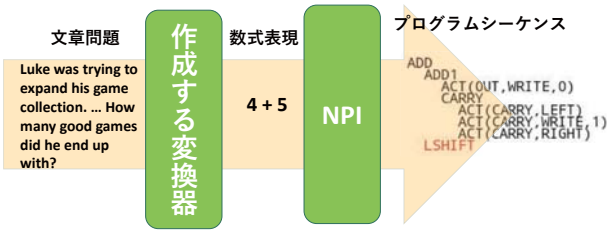

図 6: 文章問題からプログラムシーケンスまでの流れ

4.1.1 エンコーダー

エンコーダーは，アテンションレイヤーとデコーダーに、入力された文章を固定長ベクトルにして渡す．ここでは、双方向 LSTM (BiLSTM) を使用している．まず，入力単語列

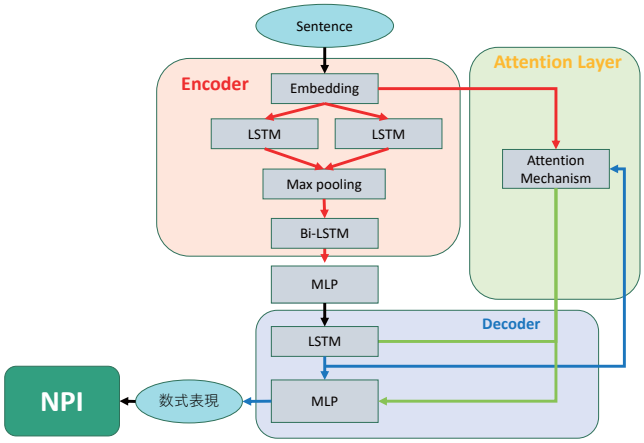

図 7: 変換器の構造

$w_1, w_2, ..., w_L$ (単語 ID) それぞれに対して学習可能な行列を使って固定長ベクトル $e(i_k)$ へと変換する．次に，それらを最初からと最後から、別々の LSTM を使って，最初からの文脈表現 $l(i_k)$ と，最後からの文脈表現 $r(i_k)$ へと式 (1) のように変換する．ここで f_l および f_r は LSTM の計算を表している．得られた $l(i_k)$ と $r(i_k)$ を一つの固定長ベクトル c_k とするため，式 (2) のようにして c_k を計算する．ここで，W_e, b_e はパラメーターであり，[;] はベクトル同士の結合を表している．最後に Bi-LSTM を使って c_k をまとめてエンコードは終了する．

$$l(i_k) = f_l(l(i_{k-1}), e(i_k)), \quad r(i_k) = f_r(r(i_{k+1}), e(i_k)) \quad (1)$$

$$c_k = tanh(W_e[max(l(i_k), r(i_k)); e(i_k)] + b_e) \quad (2)$$

4.1.2 アテンションレイヤー

アテンションレイヤーの役割は，デコーダーが入力のどこに注目すればいいかという情報をデコーダーに渡すことである．はじめに，エンコーダーから受け取った情報 c_k とデコーダーの前回の出力 h_{t-1} を式 (3) のようにして変換し，a_k^{key} と a_t^{query} を得る．それらを結合し，e_{kt} を式 (4) のように計算する．ここで，$f_{key}, f_{query}, f_{score}$ は全て，全結合層の処理を表している．最後に e_{kt} を正規化し，α_{kt} を重みとして計算した c_k の合計，att_t を式 (5) のようにして計算する．

$$a_k^{key} = f_{key}(c_k), \ a_t^{query} = f_{query}(h_{t-1}) \quad (3)$$

$$e_{kt} = f_{score}(a_k^{key}, a_t^{query}) \quad (4)$$

$$\alpha_{kt} = \frac{exp(e_{kt})}{\sum_{i=1}^{k} exp(e_{it})}, \ att_t = \sum_{i=1}^{k} \alpha_{it} c_i \quad (5)$$

4.1.3 デコーダー

デコーダーは最終的な出力系列の計算を行う．まず，アテンションレイヤーにデコーダーの前回の出力を渡し，アテンションレイヤーからの出力 att_t を受け取る．デコーダーの前回の出力 (最初の出力はエンコーダーからの出力を使う) とアテンションレイヤーからの出力をデコーダー側の LSTM に渡し，現在の出力を計算する．それと，2 層の全結合層から構成される分類器によって最終的な出力を得る．これを決められた回数繰り返し，出力系列を得る．本論文では 3 回出力している．

5. 実験

この章では前章で述べた二つのモデルに対して行った実験の説明をする．実験環境として，CPU は Intel Core i7 4790K 4.00GHz，GPU は NVIDIA GeForce GTX 950，メモリは 16GB で OS は Windows8.1 を使用した．

5.1 NPI の拡張

5.1.1 データセット

[8] によって再帰化されたタスクを行う NPI は，入出力の長さに対して完全な汎化性能を持つ．よって各ステップにおいて，起こりうる全通りのポインターの値とその出力のペアを学習データとして用いる．

5.1.2 結果

上記の学習データを学習した NPI に対して評価を行う．テストデータは各 4 つのタスクについて，入力の桁数 20 までのそれぞれ 16 個，合計 1280 個のランダムな数値のペアから作った問題を使う．入力の桁数を大きく増やしてしまうとメモリ不足になってしまい，学習ができなかった．これらのテストデータ総数に対して，正しくプログラムシーケンスを出力できた割合を NPI の評価とする．その結果を表 1 に示す．

Method	加算	減算	乗算	除算
Reed ら 2016 [10]	97%	−	−	−
Cai ら 2017 [8]	100%	−	−	−
提案手法	100%	100%	100%	100%

表 1: NPI の四則演算のテスト精度

5.1.3 サブプログラム共有の効果

表 2 にサブプログラム共有を行なった場合と行なっていない場合の比較を示す。訓練データ数は NPI の各ステップにおける入力と出力のペアの数である．学習は精度が 100%に達するか 1500epoch を超えるまで続ける．

共有しない場合では訓練データ数やプログラムリストの長さが約 2 倍必要であった．また、共有しない場合では学習を続けても精度が 100%に到達しなかった．共有した場合では 700〜800epoch の間に精度が 100%に到達することが多かった．

共有の有無	必要訓練データ数	プログラム数	精度
共有する	1,259,211 個	27 個	100%
共有しない	2,664,721 個	53 個	99.8%

表 2: サブプログラム共有の有無による性能の比較

5.2 文章題への対応

5.2.1 準備

データセットは Roy らと同じ 3 つを使用した [12].

データセット名	AI2	IL	CC
問題数	395	562	600
演算子の数	1 個または 2 個	1 個	2 個
演算子の種類	加算・減算	全て	全て
解答に関係のない数字	あり	なし	なし

表 3: データセットの特徴

比較対象となる既存手法について紹介する．Roy らによって提案された Solving General Arithmetic Word Problems (SGAWP)[12] は文章から，数字の計算順を示す配列と，計算式を表現する構文木を生成する技術である．GENERE[6]

は Bouchard らによって提案された学習データ生成モデルのパラメーターを自動的に獲得する手法である．[6] では，このGENERE と seq2seq を使って文章問題のデータセットを解いている．∂4 は Bošnjak らによって提案された Forth 言語を理解するモデルである [11]．[11] では加算，バブルソートに加えて，文章問題のデータセットを ∂4 を使って解いた．また，∂4 は AI2 データセットのような関係のない数字に対する処理を行なっておらず，計算に使う演算子も 2 個で固定されている．

5.2.2 結果

他の既存手法と比較した結果を表 4 に示す．[12] に従い，k-分割交差検証で評価する．AI2 は 3 分割，IL は 5 分割で行う．CC は [12] に従い、学習データ 300 個、バリデーションデータ 100 個、テストデータ 200 個で評価を表している．提案手法は AI2，IL において他の手法よりも精度で上回っている．しかし，CC に関しては GENERE が一番精度が良かった．

手法	AI2	IL	CC
数式表現のみ			
SGAWP [12]	82.4%	75.4%	55.5%
GENERE [6]	77.9%	56.7%	**98.5%**
seq2seq [6]	72.5%	53.7%	95.0%
解答まで			
∂4 [11]	−	−	96.0%
NPI プログラムシーケンス + 解答			
提案手法	**83.3%**	**85.1%**	97.4%

表 4: 既存手法と提案手法を比較した表

6. 考察

NPI の実験において，サブプログラムを共有して学習することは、今まで学習したことの一部を新しい学習に応用するという人間が実際に行うことをコンピューターに可能にさせているとも捉えられる．また、本研究では end-to-end 学習を行なっていない．しかし、これも人間と同様で、四則演算の計算を覚えてから文章問題に取り組むということをコンピューターに行なわせていると考えられる．コンピューターに段階的な学習をさせることの有用性は [13] でも述べられている．[13] では、あるタスクを学習させる際により簡単な訓練データから学習させることで精度が向上することを示している．

本研究ではアテンションメカニズムによって通常の seq2seq よりも性能を向上させている．例えば “Sam’s dog had puppies and 8 had spots . He gave 2 to his friends . He now has 6 puppies . How many puppies did he have to start with ?” という問題に対しては “has”や “gave” に注目していることがわかった．このような動詞から演算子の種類を判断することは非常に良いことであり、文章問題の読解にアテンションメカニズムが貢献していることがわかる．アテンションメカニズムの有効性を検証するために、提案手法のモデルからアテンションメカニズムを除いたモデルを作成し，性能の比較を行なった．その結果の表を表 5 に示す．表からわかるようにアテンションメカニズムは全てのケースにおいて良い結果を得られている．

アテンションメカニズムの存在	AI2	IL	CC
アテンションメカニズムあり	83.3%	85.1%	97.4%
アテンションメカニズムなし	71.0%	78.9%	95.9%

表 5: アテンションメカニズムの有無による性能の比較

7. まとめと今後の課題

本論文では NPI の四則演算への対応と [10] で課題として挙げられていたサブプログラム共有のための拡張を提案し、それぞれの拡張がうまく機能していることを示した．また、NPI を利用した小学生レベルの算数の文章問題のデータセットを解く手法を提案した．そして提案手法が他の先行研究よりも優れていること、アテンションメカニズムが今回使用した文章問題に有効であることを示した．

NPI の除算は商を立てる際の効率が悪い．NPI がニューラルネットワークを使用しているという利点を活かし、商を賢く立てられるようにするべきである．NPI は出力すべきプログラムの長さに対しては汎用性があるものの未知のタスクに対応できない．[2] は [1] 同様、外部メモリとコントローラーを組み合わせたモデルであり、様々なタスクに取り組んでいる．このような算術演算以外のタスクに対して NPI を適用し、プログラムを出力させるということも今後の課題としたい．

参考文献

[1] Alex Graves, Greg Wayne and Ivo Danihelka. Neural Turing Machine. *arXiv preprint arXiv:1410.5401*, 2014.

[2] Alex Graves, Greg Wayne, Malcolm Reynolds et al. Hybrid computing using a neural network with dynamic external memory. Nature, 538(7626):471-476, 2016.

[3] Chengxun Shu and Hongyu Zhang. Neural Programming by Example. In *Proc. AAAI*, pp.1539-1545, 2017.

[4] Dzmitry Bahdanaue, Kyunghyyun Cho and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] Emilio Parisotto, Adbel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyoung Zhou and Pushmeet Kohli. Neuro-Symbolic Program Synthesis. In *arXiv preprint arXiv:1611.01855*, 2016.

[6] Guillaume Bouchard, P Saito Stenetorp and Sebastian Riedel. Learning to Generate Textual Data. In *Proc. EMNLP*, pp.1608-1616, 2016.

[7] Ilya Sutskever, Oriol Vinyals and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. In *Proc. NIPS*, pp.3104-3112, 2014.

[8] Jonathon Cai, Richard Shin and Dawn Song. Making Neural Programming Architectures Generalize via Recursion. In *arXiv preprint arXiv:1704.06611*, 2017.

[9] Lieberman Henry. Your wish is my command: Programming by example. Morgan Kaufmann. 2001.

[10] Reed Scott and Nando De Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

[11] Sebastian Riedel, Matko Bošnjak and Tim Rocktäschel. Programming with a Differentiable Forth Interpreter. In *arXiv preprint arXiv:1605.06640*, 2017.

[12] Subhro Roy and Dan Roth. Solving General Arithmetic Word Problems. In *Proc. EMNLP*, pp.1743-1752, 2015.

[13] Yoshua Bengio, Jérôme Louradour, Ronan Collobert and Jason Weston. Curriculum Learning. In *Proc. ICML*, pp.41-48, 2009.